Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1-20. (Cancelled).

21. (Currently amended) ~~In a~~ A graphics system, including:

a graphics application communicating with graphics hardware through a first driver sharing an interface with the graphics application, and a second driver sharing another interface with the graphics hardware, ~~and~~ a graphics interface through which the first driver transmits original graphics call sequences to the second driver, and a performance optimization apparatus; ~~comprising:~~

wherein the performance optimization apparatus comprises at least one graphics call sequence optimizer, operationally interposed between the first and second driver, configured to process one of the original graphics call sequence to produce an optimized graphics call sequence; and

wherein the performance optimization apparatus comprises measuring means for measuring performance of the graphics system when processing the optimized graphics call sequence.

22. (Previously presented) The apparatus of claim 21, wherein the at least one graphics call sequence optimizer provides the optimized graphics call sequence directly to the graphics hardware.

23. (Previously presented) The apparatus of claim 21, wherein the at least one graphics call sequence optimizer comprising:

a first graphics call sequence optimizer that receives the original graphics call sequence and generates an intermediate optimized graphics call sequence; and

a second graphics call sequence optimizer that receives the intermediate optimized graphics call sequence and generates the optimized graphics call sequence.

24. (Previously presented) The apparatus of claim 21, further comprising:

a capture mechanism configured to capture the original graphics call sequence transmitted between the first and second drivers and to provide the captured graphics call sequence to the at least one optimization means.

25. (Previously presented) The apparatus of claim 21, wherein the at least one graphics call sequence optimizer comprises:

a graphics state call coalescer constructed and arranged to eliminate from a continuous series of graphics state calls in the original graphics call sequence one or more graphics state calls that do not cause the continuous series of graphics state calls to effect a change in a rendering by the graphics hardware.

26. (Previously presented) The apparatus of claim 21, wherein the at least one graphics call sequence optimizer comprises:

a graphics primitive coalescer constructed and arranged to coalesce a continuous series of primitive command sets occurring in the original graphics call sequence, the primitive command sets specifying a same type of discrete primitive and comprising at least one graphics vertex call, the graphics primitive coalescer generating a coalesced primitive command set comprising a plurality of

graphics vertex calls causing the graphics hardware to generate a same rendering as the continuous series of primitive command sets.

27. (Previously presented) The apparatus of claim 26, wherein the graphics interface is an OpenGL Application Program Interface (API), and wherein each the primitive command set comprises:

a glBegin()/glEnd() graphics call pair, wherein the at least one graphics vertex call is interposed between a glBegin() graphics call and a glEnd() graphics call of the glBegin()/glEnd() graphics call pair,

wherein the graphics primitive coalescer is constructed and arranged to remove all glBegin() graphics calls and glEnd() graphics calls except a glBegin() graphics call occurring first in the continuous series of primitive command sets and a glEnd() graphics call occurring last in the continuous series of primitive command sets.

28. (Previously presented) The apparatus of claim 21, wherein the graphics application is modified based upon performance improvements achieved in the graphics hardware when provided the optimized graphics call sequence, the modified graphics application producing the optimized graphics call sequence rather than the original graphics call sequence.

29. (Previously presented) The apparatus of claim 21, wherein the first driver is modified based upon performance improvements achieved in the graphics hardware when provided the optimized graphics call sequence, the modified first driver producing a new graphics call sequence including at least a portion of the optimized graphics call sequence.

30. (Previously presented) The apparatus of claim 21, wherein the apparatus is implemented in the second driver to cause the second driver, receiving the

original graphics call sequence from the first driver, to generate the optimized graphics call sequence during real-time operations of the graphics system.

31. (Previously presented) The apparatus of claim 21, wherein the at least one optimization means is implemented in the graphics hardware to cause the graphics hardware, in response to receiving the original graphics call sequence from the second driver, to generate the optimized graphics call sequence during real-time operations of the graphics system.

32. (Previously presented) The apparatus of claim 21, wherein the at least one optimization means is distributed in the second driver and the graphics hardware to generate the optimized graphics call sequence during real-time operations of the graphics system.

33. (Previously presented) The apparatus of claim 26, wherein the plurality of graphics vertex calls in the coalesced primitive command set define a strip primitive and wherein the graphics primitive coalescer is constructed and arranged to remove redundant graphics vertex calls from the coalesced primitive command set that define vertices common to neighboring primitives of the strip primitive and to alter a primitive type specified by the coalesced primitive command set to identify the primitive strip rather than the discrete primitive type.

34. (Previously presented) The apparatus of claim 21, wherein the at least one graphics call sequence optimizer comprises:

a common state primitive compiler constructed and arranged to prevent frequent toggling of graphics state values in the graphics hardware caused by the original graphics call sequence.

35. (Previously presented) The apparatus of claim 34, wherein the common state primitive compiler compiles primitive command sets in the original graphics primitive sequence in accordance with common graphics state values in which

they are to be rendered, and to generate one or more sequences of the compiled primitive command sets preceded by graphics state calls necessary to establish the common graphics state values.

36. (Previously presented) The apparatus of claim 21, wherein the at least one graphics call sequence optimizer comprises:

a primitive type converter constructed and arranged to convert a primitive type specified in a primitive command set in the original graphics call sequence from a noncombinable primitive type including primitives that cannot be coalesced by the graphics primitive coalescer to a combinable primitive type that can be coalesced by the graphics primitive coalescer.

37. (Previously presented) The apparatus of claim 21, wherein the at least one graphics call sequence optimizer comprises:

a vertex array builder constructed and arranged to create a vertex array having vertices identified in a series of graphics vertex calls of a primitive command set in the original graphics call sequence, and to generate an associated offset for reference by a graphics array call that uses the array of vertices to render a number of specified primitives.

38. (Previously presented) The apparatus of claim 21, wherein the at least one graphics call sequence optimizer comprises:

a vertex loop generator constructed and arranged to generate a repeatable loop to efficiently process repetitive series of graphics calls occurring in a primitive command set in the original graphics call sequence.

39. (Previously presented) The apparatus of claim 21, wherein the measuring means comprises:

means for measuring performance of the original graphics call sequence to obtain a performance baseline against which the performance of the optimized graphics call sequence can be compared.

40. (Previously presented) The apparatus of claim 21, wherein the measuring means comprises:

means for compiling the original and the optimized graphics call sequences.

41. (Previously presented) In a graphics system including a graphics application communicating with graphics hardware through a first driver interfaced with the graphics application and a second driver interfaced with the graphics hardware, and a graphics interface through which the first driver transmits original graphics call sequences to the second driver, a method for optimizing the original graphics call sequence to generate an optimized graphics call sequence causing the graphics hardware to render with improved performance, a same image as the original graphics call sequence, the method comprising:

capturing the original graphics call sequence;

restructuring the original graphics call sequence to produce the optimized graphics call sequence; and

measuring performance of the graphics system when processing the optimized graphics call sequence.

42. (Previously presented) The method of claim 41, wherein restructuring the original graphics call sequence comprises:

removing from a continuous series of graphics state calls in the original graphics state sequence all redundant, duplicative and otherwise unnecessary graphics state calls to form a coalesced continuous series of graphics state calls, wherein the coalesced continuous

series of graphics state calls includes a number of graphics state calls that is less than a number of graphics state calls in the continuous series of graphics state calls.

43. (Previously presented) The method of claim 42, wherein the graphics state calls include multiple first graphics state calls setting values of a first graphics state and wherein the removing operation comprises:

removing all occurrences of the first graphics state calls except a last occurring first graphics state call to form a first reduced graphics state call sequence; and

removing from the first reduced graphics state call sequence all graphics state calls that set a graphics state to a value that the graphics state is currently set at in the graphics hardware.

44. (Previously presented) The method of claim 41, wherein restructuring the original graphics call sequence comprises:

coalescing graphics vertex calls contained in a continuous series of primitive command sets that render primitives of the same type in the original graphics call sequence to generate a corresponding coalesced primitive command set in the optimized graphics call sequence effecting a same rendering in the graphics hardware as the original graphics call sequence.

45. (Previously presented) The method of claim 44, wherein the graphics application program interface is an OpenGL graphics API, and wherein coalescing graphics vertex calls comprises:

removing intermediate glBegin()/glEnd() pairs of the primitive command sets to create a single primitive command set comprising all of the graphics vertex calls in the original continuous series of primitive command sets.

46. (Previously presented) The method of claim 45, wherein coalescing graphics vertex calls further comprises:

determining whether the graphics primitive calls in the single primitive command set define discrete primitives having sufficient common vertices to be rendered as a primitive strip;

eliminating, when a strip primitive is possible, redundant graphics vertex calls; and

changing, when a strip primitive is possible, the specified primitive type to an equivalent primitive strip primitive type.

47. (Previously presented) The method of claim 41, wherein restructuring the original graphics call sequence comprises:

compiling a continuous series of primitive command sets that are to be rendered with a common graphics state values preceded by graphics state calls necessary to establish the common graphics state values in which the primitive command sets are to be rendered.

48. (Previously presented) The method of claim 47, wherein the graphics application program interface is an OpenGL graphics API, and wherein compiling a continuous series of primitive command sets comprises:

compiling the primitive command sets based upon graphics state calls immediately preceding the primitive command sets; and

associating only graphics state calls necessary for rendering each the compiled list of primitive command sets with the compiled list of the primitive command sets.

49. (Previously presented) The method of claim 41, wherein restructuring the original graphics call sequence comprises:

converting, prior to compiling a continuous series of primitive command sets, the specified primitive type from a primitive type that cannot be coalesced to a primitive type that can be coalesced.

50. (Previously presented) The method of claim 49, wherein the graphics API is an OpenGL graphics API, and wherein the graphics primitives comprise line strips, triangle strips, quadrilateral strips, triangle fans, line loops and polygon primitive types.

51. (Previously presented) The method of claim 49, wherein the graphics API is an OpenGL API, and wherein converting the specified primitive type comprises:

converting a primitive type specified in a primitive command set from a line strip to a line when the primitive command set includes 2 graphics vertex calls.

52. (Previously presented) The method of claim 49, wherein the graphics interface is an OpenGL API, and wherein converting the specified primitive type comprises:

converting a primitive type specified in a primitive command set from a triangle strip to a triangle when the primitive command set includes 3 graphics vertex calls.

53. (Previously presented) The method of claim 49, wherein the graphics interface is an OpenGL API, and wherein converting the specified primitive type comprises:

converting a primitive type specified in a primitive command set from a quadrilateral strip to a quadrilateral when the primitive command set includes 4 graphics vertex calls.

54. (Previously presented) The method of claim 49, wherein the graphics interface is an OpenGL API, and wherein converting the specified primitive type comprises:

converting a primitive type specified in a primitive command set from a triangle fan to a triangle when the primitive command set includes 3 graphics vertex calls.

55. (Previously presented) The method of claim 49, wherein the graphics interface is an OpenGL API, and wherein converting the specified primitive type comprises:

converting a primitive type specified in a primitive command set from a line loop to a line when the primitive command set includes 2 graphics vertex calls.

56. (Previously presented) The method of claim 49, wherein the graphics interface is an OpenGL API, and wherein converting the specified primitive type comprises:

converting a primitive type specified in a primitive command set from a polygon strip to a quadrilateral when the primitive command set includes 4 graphics vertex calls.

57. (Previously presented) The method of claim 41, wherein restructuring the original graphics call sequence comprises:

creating a vertex array having vertices identified in a series of graphics vertex calls of a primitive command set for reference by an associated pointer and graphics array call to render one or more of a specified primitive.

58. (Previously presented) The method of claim 57, wherein compiling a continuous series of primitive command sets comprises:

creating a vertex array having vertices identified in a series of graphics vertex calls of a primitive command set; and

generating the associated pointer and the graphics array call referencing the pointer.

59. (Previously presented) The method of claim 58, wherein creating a vertex array having vertices identified in a series of graphics vertex calls of a primitive command set comprises:

accumulating into an array a plurality of vertices specified in all graphics vertex calls occurring in a primitive command set; and

wherein generating the associated pointer and the graphics array call referencing the pointer comprises:

generating a pointer to the array; and

generating an appropriate array processing graphics call that accesses the vertex array using the pointer,

wherein the array processing graphics call accesses the array, causing a rendering by the graphics hardware of a graphics primitive having its vertices located at the locations stored in the array.

60. (Previously presented) The method of claim 58, wherein the graphics interface is an OpenGL API, and wherein the graphics call is glDrawArrays graphics call and the pointer is defined by a glVertexPointer graphics call.

61. (Previously presented) The method of claim 41, wherein restructuring the captured graphics call sequence comprises:

creating a vertex array having vertices identified in a series of graphics vertex calls of a primitive command set in the original graphics call sequence for reference by an associated graphics array call to render one or more specified primitives.

62. (Previously presented) The method of claim 61, wherein compiling a continuous series of primitive command sets comprises:

identifying graphics calls that comprise a repetitive series;

counting a number of occurrences each the graphics call continuously appears in the primitive command set; and

generating an optimized graphics call sequence effecting a loop execution of the repetitive graphics calls for the identified number of times the particular repetitive pattern should be implemented.

63. (Previously presented) The method of claim 41, wherein restructuring the original graphics call sequence comprises:

removing from a continuous series of graphics state calls in the original graphics state sequence all redundant, duplicative and otherwise unnecessary graphics state calls to form a coalesced continuous series of graphics state calls, wherein the coalesced continuous series of graphics state calls includes a number of graphics state calls less than a number of graphics state calls in the continuous series of graphics state calls and causing the graphics hardware to render a same image as the continuous series of graphics state calls.

64. (Previously presented) The method of claim 61, wherein restructuring the original graphics call sequence comprises:

compiling a continuous series of primitive command sets that are to be rendered with a same state values preceded by graphics state calls necessary to establish common state values in which the primitive command sets are to be rendered.

65. (Previously presented) The method of claim 41, wherein measuring a performance of the optimized graphics call sequence comprises:

measuring a performance of the original graphics call sequence to obtain a performance baseline against which the performance of the optimized graphics call sequence can be compared.

66. (Previously presented) The method of claim 41, wherein measuring a performance of the optimized graphics call sequence comprises:

compiling the original and the optimized graphics call sequence.

67. (Previously presented) In a graphics system including a graphics application communicably coupled with a graphics hardware through a first driver interfaced with the graphics application and a second driver interfaced with the graphics hardware, the drivers communicating with each other through a graphics interface, a performance optimization apparatus comprising:

graphics call sequence optimization means for processing the original graphics call sequence to produce an optimized graphics call sequence; and

measuring means for measuring performance of the graphics system when processing the optimized graphics call sequence.

68. (Previously presented) The apparatus of claim 67, comprising:

capture means, operatively coupled to a communications path coupling the first and second drivers, for capturing the original graphics call sequence transmitted between the first and second drivers; and

means for providing the captured graphics call sequence to the at least one graphics call sequence optimization means.

69. (Previously presented) The apparatus of claim 67, wherein the at least one optimization means comprises:

graphics state call coalescing means for eliminating from a continuous series of graphics state calls in the original graphics call sequence one or more graphics state calls that do not cause the continuous series of graphics state calls to effect a change in a rendering by the graphics hardware.

70. (Previously presented) The apparatus of claim 67, wherein the at least one optimization means comprises:

graphics primitive coalescing means for coalescing a continuous series of primitive command sets occurring in the captured graphics state call, the primitive command sets specifying a same type of discrete primitive and comprising graphics vertex call, the graphics primitive coalescer generating a coalesced primitive command set comprising a plurality of graphics vertex calls causing the graphics hardware to generate a same rendering as the continuous series of primitive command sets.

71. (Previously presented) The apparatus of claim 70, wherein the graphics interface is an OpenGL Application Program Interface (API), and wherein each the primitive command set comprises:

a glBegin()/glEnd() graphics call pair, wherein the at least one graphics vertex call is interposed between a glBegin() graphics call and a glEnd() graphics call of the glBegin()/glEnd() graphics call pair,

wherein the graphics primitive coalescing means removes all glBegin() graphics calls and glEnd() graphics calls except a @Begin() graphics call occurring first in the continuous series of primitive command sets and a glEnd() graphics call occurring last in the continuous series of primitive command sets.

72. (Canceled).

73. (Previously presented) The apparatus of claim 67, wherein the optimization means is implemented in the first driver.

74. (Previously presented) The apparatus of claim 67, wherein the optimization means is implemented in the second driver.

75. (Previously presented) The apparatus of claim 67, wherein the optimization means is implemented in the graphics hardware.

76. (Previously presented) The apparatus of claim 70, wherein the plurality of graphics vertex calls in the coalesced primitive command set define a strip primitive and wherein the graphics primitive coalescing means comprises:

means for removing redundant graphics vertex calls from the coalesced primitive command set that define vertices common to neighboring primitives of the strip primitive and to alter a primitive type specified by the coalesced primitive command set to identify the primitive strip rather than the discrete primitive type.

77. (Previously presented) The apparatus of claim 67, wherein the optimization means comprises:

state compiling means for compiling primitive command sets in the original graphics primitive sequence in accordance with graphics state values in which they are to be rendered and for generating one or more sequences of the compiled primitive command sets preceded by graphics state calls necessary to establish a common graphics state values.

78. (Previously presented) The apparatus of claim 69, wherein the optimization means further comprises:

primitive type converting means for converting a primitive type specified in a primitive command set in the original graphics call sequence from a non-combinable primitive type including primitives that cannot be coalesced by the graphics primitive coalescing means to a combinable primitive type that can be coalesced by the graphics primitive coalescing means.

79. (Previously presented) The apparatus of claim 67, wherein the optimization means comprises:

vertex array generating means for creating a vertex array having vertices identified in a series of graphics vertex calls of a primitive command set in the original graphics call sequence, and to generate an associated offset for reference by a graphics array call that uses the array of vertices to render a number of specified primitives.

80. (Previously presented) The apparatus of claim 67, wherein the optimization means comprises:

vertex loop generating means for generating a repeatable loop to efficiently process repetitive series of graphics calls occurring in a primitive command set in the original graphics call sequence.

81. (Previously presented) The apparatus of claim 67, wherein the measuring means comprises:

means for measuring performance of the original graphics call sequence to obtain a performance baseline against which the performance of the optimized graphics call sequence can be compared.

82. (Previously presented) The apparatus of claim 67, wherein the measuring means comprises:

means for compiling the original and the optimized graphics call sequence.

83. (Previously presented) A computer program product for use with a computing system having at least one central processing unit (CPU) and an operating system and operatively coupled to a graphics system including a graphics application communicating with graphics hardware through a first driver interfaced directly with the graphics application and a second driver interfaced directly with the graphics hardware, and a graphics interface through which the first driver transmits original graphics call sequences to the second driver, the computer program product comprising a computer usable medium having embodied therein computer readable program code method steps, the method steps comprising:

capturing the original graphics call sequence;

restructuring the original graphics call sequence to produce the optimized graphics call sequence; and

measuring performance of the graphics system when processing the optimized graphics call sequence.

84. (Previously presented) The computer program product of claim 83, wherein restructuring the original graphics call sequence comprises:

removing from a continuous series of graphics state calls in the original graphics state sequence all redundant, duplicative and otherwise unnecessary graphics state calls to form a coalesced continuous series of graphics state calls, wherein the coalesced continuous series of graphics state calls includes a number of graphics state calls that is less than a number of graphics state calls in the continuous series of graphics state calls.

85. (Previously presented) The computer program product of claim 83, wherein restructuring the original graphics call sequence comprises:

coalescing graphics vertex calls contained in a continuous series of primitive command sets that render primitives of the same type in the original graphics call sequence to generate a corresponding coalesced primitive command set in the optimized graphics call sequence effecting a same rendering in the graphics hardware as the original graphics call sequence.

86. (Previously presented) The computer program product of claim 83, wherein restructuring the original graphics call sequence comprises:

compiling a continuous series of primitive command sets that are to be rendered with necessary to establish the common graphics state values in which the primitive command sets are to be rendered.

87. (Previously presented) The computer program product of claim 83, wherein the restructuring the original graphics call sequence comprises:

creating a vertex array having vertices identified in a series of graphics vertex calls of a primitive command set for reference by an associated pointer and graphics array call to render one or more of a specified primitive.

88. (Previously presented) The computer program product of claim 83, wherein measuring a performance of the optimized graphics call sequence comprises:

measuring a performance of the original graphics call sequence to obtain a performance baseline against which the performance of the optimized graphics call sequence can be compared.

89. (Previously presented) The computer program product of claim 83, wherein measuring a performance of the optimized graphics call sequence comprises:

compiling the original and the optimized graphics call sequence.